Adapting and Extending Token-Aware Virtual Adversarial Training on the Named Entity Recognition Problem

Aaron Reich
Georgia Tech
Atlanta, GA
areich8@gatech.edu

Isha Shah
Georgia Tech
Atlanta, GA
isha.shah@gatech.edu

Pradeepti Gupta
Georgia Tech
Atlanta, GA
pgupta339@gatech.edu

Sanghavi Gaddam
Georgia Tech
Atlanta, GA
sanghavig@gatech.edu

Abstract

Adversarial training techniques have been shown to increase robustness and generalization. However gradient-based adversarial training techniques used in computer vision cannot be directly applied to language because of the discreteness of the data. An adversarial training method called FreeLB (Free Large-Batch) has been introduced to the NLP field and involves the addition of perturbations to word embeddings and the minimization of the adversarial loss around the input examples. Token-Aware Virtual Adversarial Training (TAVAT) further improves upon this approach by making use of a token-level normalization ball and perturbation vocabulary allowing for token level information to be incorporated into gradient-based virtual adversarial training approaches. Named Entity Recognition (NER) is an important problem in NLP involving the assigning of named entity labels to words in text. We adapt TAVAT for the NER problem and make different alterations to it with the purpose of empirically finding promising directions in virtual adversarial training to improve the generalization of deep learning models.

1. Introduction

Deep learning models have achieved great performance on many NLP problems [2, 1], but have been shown to widely misclassify examples that are perturbed only slightly to be different from the original examples [3]. These perturbed examples are referred to as adversarial examples. Adversarial training has been shown to increase robustness against adversarial examples and increase generalization performance on clean examples (non-perturbed) as well.

Named Entity Recognition (NER) is an important problem in NLP involving the assigning of named entity labels to words in text. It is a sequence labeling problem. Sequence labeling is a type of pattern recognition task that involves the algorithmic assignment of a categorical label to each member of a sequence of observed values. A common example of a sequence labeling task is part of speech tagging, which seeks to assign a part of speech to each word in an input sentence or document. Sequence labeling can be treated as a set of independent classification tasks, one per member of the sequence. However, accuracy is generally improved by making the optimal label for a given element dependent on the choices of nearby elements, using special algorithms to choose the globally best set of labels for the entire sequence at once. In short, it can be described as to algorithmically map a sequence on one alphabet to a "good" sequence on another alphabet. Two forms of sequence labeling are: Token Labeling: Each token gets an individual Part of Speech (POS) label and Span Labeling: Labeling segments or groups of words that contain one tag (Named Entity Recognition, Syntactic Chunks).

There are many works that explore the use of the construction of adversarial examples at the text-level [6]. However the popular gradient-based adversarial training techniques that are applied widely in computer vision cannot be directly applied to language because the data is discrete. So alternatively virtual adversarial training techniques are used [4].

A virtual adversarial training method called FreeLB (Free Large-Batch) was proposed by [8], which involves the addition of perturbations to word embeddings and the minimization of the adversarial loss around the input examples. It makes use of "free" training techniques proposed

in the literature which alter the training set with diverse adversarial examples making use of various norm constraints. It does this at no extra cost than the PGD-based (Projected Gradient Descent) adversarial training method and also leads to improved invariance in the embedding space which is correlated with better generalization.

All of the virtual adversarial training methods prior to Token-Aware Virtual Adversarial Training(TAVAT) [4] involve the generation of perturbations constrained by Frobenius normalization balls. TAVAT instead involves the generation of fine-grained perturbations, and makes use of a token-level accumulated perturbation vocabulary for initializing the perturbations better and makes use of a token-level normalization ball for constraining the perturbations more precisely.

We are interested mainly in the overall increase to generalization that virtual adversarial training provides to NER models, and what directions are promising to further increase generalization. So we adapt the TAVAT for the NER problem and empirically study the results from different alterations to it.

2. Approach

2.1. Adapting TAVAT for the NER Problem

The GitHub repository [1] corresponding to TAVAT [4] did not contain code for usage on the NER problem. The NER dataset we use is CoNLL 2003 [5] with the BIO labeling scheme. Methods such the ones for reading in the data, converting the data to features, and evaluation had to be incorporated into the current code base. The classifier model used for this task was BertForTokenClassification which utilizes the BERT Base model with 12 layers and 110 million trainable parameters. In order to convert the data to features, we use a BERT Tokenizer to convert the words into IDs to be used for retrieving the corresponding BERT embeddings. We map the labels to IDs as well. The output of the model is the logits for each label, and the label id corresponding to the maximum value is mapped to the predicted token label. For the evaluation method, the NER evaluation metric we use is the one proposed for the CoNLL task with the precision, recall, and F1-score being the computed scores. We used cross entropy loss as our loss function and AdamW as our optimizer. Other methods such as train and main had to be updated as well.

2.2. Hyperparameter Tuning

To understand the impact of different hyperparameters on the performance of our model, hyperparameters such as adversarial training (K), adversarial step size (α), and perturbation constrain bound (ϵ) are tuned. The model is trained using different values of one hyperparameter while keeping all other hyperparameters constant. This helps in studying the effect of one hyperparameter on the overall model performance.

[1] https://github.com/LinyangLee/Token-Aware-VAT

2.3. Initialization of Token-Level Perturbation

Generally, adversarial training involves randomly initializing perturbations in every mini batch. This works out well for images since pixels do not contain similar information across different images. However, similar information is carried by tokens in languages across different sequences and randomly initializing the perturbations could cause unnecessary noise to be introduced during the training process. To tackle this problem, we create a Global Perturbation Vocabulary so that perturbations of the same token may be accumulated in one place and then be used for initialization of perturbations. We perform a comparative study on randomly initializing perturbations and initializing perturbations from the Global Perturbation Vocabulary to understand the impact of both on the training process.

2.4. Removing Token-Level Constraint in the Norm

As we use gradients to update perturbations and then restrict them within normalized limits, to keep our perturbations to the least. These perturbations are different from the instance level perturbations by a token level constraints as the former act on the embedding space which is at token level. As different tokens have varying impact in a sequence, some are super critical while others are not, hence there were varying bounds for gradients with higher values over the lower ones. This was achieved using the scaling index which normalizes the perturbations over separate tokens accordingly. After scaling normalization with the token level constraint Frobenius normalization is performed.

2.5. Learning the Function for Combining the Embeddings and Pertubations

In Figure 1 line 6, it can be observed how in the original TAVAT algorithm, the instance-level pertubation δ_{t-1} and token-level pertubation η_{t-1} are both added to the data sample X before being fed to the model. Let parameter θ be a learned parameter and S be the standard Softmax function:

$$[\hat{\theta_1}\hat{\theta_2}] = S([\theta_1\theta_2]) \tag{1}$$

Line 6:

$$X + \delta_{t-1} + \eta_{t-1} \tag{2}$$

can be replaced with:

$$X + \hat{\theta_1}\delta_{t-1} + \hat{\theta_2}\eta_{t-1} \tag{3}$$

Algorithm 1 Token-Aware Virtual Adversarial Training

Require: Training Samples $S = \{(X = [w_0, \cdots, w_i, \cdots], y)\}$, perturbation bound ϵ, initialize bound σ adversarial steps K, adversarial step size α, model parameter θ

$\boldsymbol{V} \in \mathbb{R}^{N \times D} \leftarrow \frac{1}{\sqrt{D}} U(-\sigma, \sigma)$ // Initialize perturbation vocabulary $\boldsymbol{V}$
for epoch = $1, \cdots,$ **do**
  for batch $B \subset S$ **do**
    $\delta_0 \leftarrow \frac{1}{\sqrt{D}} U(-\sigma, \sigma)$, $\eta_0^i \leftarrow \boldsymbol{V}[w_i]$, $g_0 \leftarrow 0$ //Initialize perturbation and gradient of θ
    for t = $1, \cdots, K$ **do**
      $g_t \leftarrow g_{t-1} + \frac{1}{K} \mathbb{E}_{(X,y) \in B} [\bigtriangledown_\theta L(f_\theta(X + \delta_{t-1} + \eta_{t-1}), y)]$ //Accumulate gradients of θ
      Update token-level perturbation η:
      $g_\eta^i \leftarrow \bigtriangledown_{\eta^i} L(f_\theta((X + \delta_{t-1} + \eta_{t-1}), y)$
      $\eta_t^i \leftarrow n^i * (\eta_{t-1}^i + \alpha \cdot g_\eta^i / ||g_\eta^i||_F)$
      $\eta_t \leftarrow \prod_{||\eta||_F < \epsilon} (\eta_t)$
      Update instance-level perturbation δ:
      $g_\delta \leftarrow \bigtriangledown_\delta L(f_\theta((X + \delta_{t-1} + \eta_{t-1}), y)$
      $\delta_t \leftarrow \prod_{||\delta||_F < \epsilon} (\delta_{t-1} + \alpha \cdot g_\delta / ||g_\delta||_F)$
    end for
    $\boldsymbol{V}[w_i] \leftarrow \eta_K^i$ //Update perturbation vocabulary $\boldsymbol{V}$
    $\theta \leftarrow \theta - g_K$ //Update model parameter θ
  end for
end for

Figure 1: TAVAT Algorithm Pseudocode

This allows for an interpolation of δ_{t-1} and η_{t-1} by the parameter θ which could allow the model to learn to use more of δ_{t-1} or η_{t-1} depending the situation.

2.6. Experimenting with Nuclear Norm for Standardization in place of Frobenius Norm

As specified in the above section, token level constraints are used as scaling indexes to normalize each type of gradients and then Frobenius norm is used. Instead of using Frobenius norm, we are exploring nuclear norm to standardize the gradients.

The nuclear norm (sometimes called Schatten 1 -norm or trace norm) of a matrix A , is defined as the sum of its singular values. The norm can be computed from the singular value decomposition of A.

2.7. Applying Noise to the Word Embeddings and Hidden Layers

Our next attempt at improving TAVAT was to introduce Gaussian noise and Bernoulli noise to the word embeddings. [7] explores the effects of adding Gaussian noise, Bernoulli noise, and adversarial noise to the input layer embeddings. Gaussian and Bernoulli noise are both forms of random noise that are added without domain knowledge. Gaussian noise use a Gaussian distribution to randomly add small amounts of noise and Bernoulli noise is essentially the same as dropout where some inputs are dropped entirely based on a certain probability. On the other hand, adversarial noise involves using domain knowledge to make modifications such as replacing words with their synonyms [7].

In the experiments [7] conducted, they found that often Gaussian noise performed better than Bernoulli noise since dropout might be too aggressive of an approach if the model still needs certain words to be present to learn properly. Our intention with this experiment is to test if the added noise can make the model more generalizable when it is evaluated against the test set.

3. Experiments and Results

We implement our model using PyTorch and Huggingface BERT. The model is trained using GEFORCE RTX 2080.

3.1. Hyperparameter Tuning

The model was trained using different values of adversarial steps (K). For K=1, the model did not perform very well. This indicates that a low value of K causes very little perturbation to be introduced into the embeddings leading to little to no improvement in the robustness of the model. For K=3 too, the model did not perform very well. This indicates that a high value of K causes large amounts of perturbations to be introduced into the embeddings, thereby shifting the perturbed embeddings far away from the original embeddings. The model does not perform well in this

case because it is unable to minimize the reduction in accuracy. The best balance between improving robustness and minimizing reduction in accuracy is achieved for K=2. The results are shown in Table 1.

Adversarial Steps (K)	F1	Precision	Recall
1	95.26	95.07	95.46
2	95.72	95.57	95.86
3	95.69	95.55	95.83

Table 1: Evaluation results on validation set for different values of adversarial steps.

The model was trained using different values of adversarial step size (α). The model performance is better for low values of adversarial step size as compared to higher values of step size, as can be seen in Table Table 2. This indicates that for higher values of α, the model encounters the issue of stale gradient. In this, there is one descent step on model parameters and simultaneously K ascent steps on the perturbation. The update of model parameters is based on gradient of loss with respect to model parameters. It may not minimize adversarial risk and so the model does not become robust. The update of perturbation is based on gradient of loss with respect to perturbation and so it may not maximize the model accuracy.

Adversarial Step Size (α)	F1	Precision	Recall
5e-2	95.72	95.57	95.86
1e-1	95.67	95.44	95.89
5e-1	95.61	95.31	95.91

Table 2: Evaluation results on validation set for different values of adversarial step size.

The model was trained using different values of perturbation bound (ϵ). As seen in Table 3, the best model performance was achieved for an intermediate value of ϵ. Lower value of ϵ implies less distance between original embeddings and perturbed embeddings. This results in a model which is not robust. Higher value of ϵ implies more distance between original embeddings and perturbed embeddings. This in turn results in a model whose accuracy has been compromised.

Perturbation Bound (ϵ)	F1	Precision	Recall
4e-1	95.61	95.39	95.83
5e-1	95.65	95.46	95.84
6e-1	95.58	95.30	95.86

Table 3: Evaluation results on validation set for different values of perturbation bound.

We were unable to achieve the same results as [4]. Table 4 shows the performance of our model after hyperparameter tuning. This model has been used as the baseline model for all our experiments.

Dataset	F1	Precision	Recall
Validation Set	95.72	95.57	95.86
Test Set	91.82	91.79	91.85

Table 4: Performance of the model on validation set and test set after hyperparameter tuning.

3.2. Effect of Noise on the Word Embeddings

Our first attempt at incorporating Gaussian noise involved adding noise to the input embeddings. For the Gaussian noise distribution, the mean was maintained at 0 so that we could ensure that the noise did not affect all of the input embeddings. As you can see from the results in Table 5, larger values of noise led to the poorest performance results. The only improvement we saw over the baseline results was in increase in the recall score with an SD value of 0.01. This indicates that adding Gaussian noise to the input embeddings might not be the best approach for our task.

SD Value	F1	Precision	Recall
0.01	91.74	91.49	92.00
0.05	91.61	91.27	91.96
0.1	89.80	89.18	90.43

Table 5: Evaluation results for different SD values of added Gaussian noise to input embeddings.

The next type of noise we attempted to apply was Bernoulli noise on the input embedding. The results for this experiment are visible in Table 6. For this noise application, the input embeddings were passed through a dropout layer. We varied the value p which is the probability of an embedding value being zeroed out. [7] mentioned the Bernoulli noise is not the most effective, however, we did see a slight increase in the F1 score with p = 0.3.

p	F1	Precision	Recall
0.1	91.21	90.88	91.55
0.3	91.87	91.62	92.12
0.5	91.44	91.27	91.61

Table 6: Evaluation results for different p values of added Bernoulli noise to input embeddings.

3.3. Initialization of Token-Level Perturbation

We experiment with different methods of initializing the token level perturbation. As seen in Table 7, the model shows better performance when token level perturbations are initialized from a Global Perturbation Vocabulary than when they are initialized randomly. This indicates that since tokens in languages carry similar information in different sequences, initializing perturbations for them from a Global Perturbation Vocabulary helps retain useful information. Furthermore, randomly initializing perturbations for tokens for different sequences introduces unnecessary noise and adds to loss of important information.

Initializing Perturbation	F1	Precision	Recall
Perturbation Vocabulary	95.72	95.57	95.86
Randomly	95.63	95.47	95.79

Table 7: Evaluation results for different methods of initializing the token-level perturbation.

3.4. Removing Token-Level Constraint in the Norm

As we use gradients to update perturbations and then restrict them within normalized limits, to keep our perturbations to the least. These perturbations are different from the instance level perturbations by a token level constraints as the former act on the embedding space which is at token level. As different tokens have varying impact in a sequence, some are super critical while others are not, hence there were varying bounds for gradients with higher values over the lower ones. This was achieved using the scaling index which normalizes the perturbations over separate tokens accordingly. After scaling normalization with the token level constraint Frobenius normalization is performed. We evaluate the performance of the impact of having Token level constraint in the normalization step and perform the experiments with and without it. Below is the summary of the evaluation of the experiments performed.

Token constraint	F1	Precision	Recall
Present	91.82	91.79	91.85
Removed	91.36	90.93	91.78

Table 8: Evaluation results for with and without Token level constraint.

This implies that adding the token level constraint not only contributed to better accuracy scores but also higher precision and recall ones. Thus, having token level constraints in the norm has high impact.

3.5. Nuclear Norm for Standardization in place of Frobenius Norm

As specified in the above section, token level constraints are used as scaling indexes to normalize each type of gradients and then Frobenius norm is used. Instead of using Frobenius norm, we are exploring nuclear norm to standardize the gradients.

The nuclear norm (sometimes called Schatten 1 -norm or trace norm) of a matrix A , is defined as the sum of its singular values. The norm can be computed from the singular value decomposition of A.

While experimenting with the nuclear norm, we ran into memory issue, 'CUBLAS error: memory mapping error (11) in magma ssytrd2 gpu'. This issue is still open yet and we are not yet able to find any resolution to overcome this memory issue. https://github.com/pytorch/pytorch/issues/4634 We are still exploring methods to overcome this memory issue by breaking into blocks and trying to run the experiment.

3.6. Learning the Function for Combining the Embeddings and Pertubations

We experimented with a θ parameter for every hidden dimension, a θ parameter for every token in the sequence, and both and the results can be seen in Table 9. The F1 score has minor increases as the number of θ parameters is increased. When inspecting the values of θ learned for every token after applying Softmax, the max value of $\hat{\theta_1}$ for δ is 0.5820, min is 0.5. For $\hat{\theta_2}$ for η, the max value is 0.5, min is 0.4180. When inspecting the values of θ learned for every hidden dimension after applying Softmax, the max value of $\hat{\theta_1}$ for δ is 0.5992, min is 0.5559. For $\hat{\theta_2}$ for η, the max value is 0.4441, min is 0.4008. The model appears to have learned to interpolate a larger portion of δ with a smaller amount of η to then be added to X. The learning of this function did however decrease performance slightly. So learning a non-linear function may need to be explored in the future to see if it will actually improve generalization.

Amount of θ parameter	F1	Precision	Recall
No use of θ	91.82	91.79	91.85
For every token	91.10	90.66	91.55
For every hidden dim.	91.37	91.04	91.71
Every token and hidden dim.	91.49	91.26	91.71

Table 9: Learning the Function for Combining the Embeddings and Pertubations. The Max sequence length is set to 256 and the number of hidden dimensions is set to 768

4. Conclusion

Our experiments on hyperparameter tuning help us conclude that the best model performance is achieved for K=2, α = 5e-2 and ϵ = 5e-1. This is because for these hyperparameter values, the perturbation introduced is just enough to improve the model robustness against adversarial attacks and at the same time minimize the reduction in accuracy.

We observed in the Removing Token-Level Constraint in the Norm experiment that adding the token level constraint contributed not only to better accuracy scores but also led to higher precision and recall values. Hence, token level constraints should be present in the norm. Our experiments with adding Gaussian and Bernoulli noise to the input word embedding revealed no significant improvements in the generalization capabilities of the model. An area for further exploration would be to consider noise that is more domain specific rather that using noise methods that do not account for the domain.

From our experiments on initialization of token level perturbations, it can be concluded that initialization using a Global Perturbation Vocabulary helps retain useful information. This helps improve the model performance. Random initialization on the other hand, introduces unnecessary noise in the training process and this hampers the model performance. The learning of the function for combining the embeddings and pertubations did decrease performance slightly. However, if a more complicated function can be learned which does help increase generalization, the interpretability of the values of the θ parameter after the softmax function may help to understand which perturbations are more important to the model.

We succeeded in adapting TAVAT to the NER problem, however our extensions to TAVAT did not provide significant improvements. We did however explore different directions for which virtual adversarial learning can be improved upon in the NLP domain.

References

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2016.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[3] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples, 2015.

[4] Linyang Li and Xipeng Qiu. Tavat: Token-aware virtual adversarial training for language understanding, 2020.

[5] Erik F. Tjong Kim Sang and Fien De Meulder. Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, pages 142–147, 2003.

[6] Boxin Wang, Chejian Xu, Shuohang Wang, Zhe Gan, Yu Cheng, Jianfeng Gao, Ahmed Hassan Awadallah, and Bo Li. Adversarial GLUE: A multi-task benchmark for robustness evaluation of language models. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021.

[7] Dongxu Zhang and Zhichao Yang. Word embedding perturbation for sentence classification, 2018.

[8] Chen Zhu, Yu Cheng, Zhe Gan, Siqi Sun, Tom Goldstein, and Jingjing Liu. Freelb: Enhanced adversarial training for natural language understanding, 2020.

A. Reproducibility Checklist

A.1. Specification of all dependencies:

- boto3 1.16.60
- botocore 1.19.60
- certifi 2020.12.5
- chardet 4.0.0
- click 7.1.2
- cycler 0.10.0
- filelock 3.0.12
- idna 2.10
- importlib-metadata 3.10.0
- jmespath 0.10.0
- joblib 1.0.1
- kiwisolver 1.3.1
- matplotlib 3.3.4
- mkl-fft 1.3.0
- mkl-random 1.1.1
- mkl-service 2.3.0
- numpy 1.19.2
- olefile 0.46
- packaging 20.9
- pandas 1.2.2
- Pillow 8.2.0
- pip 21.0.1
- protobuf 3.14.0
- pyparsing 2.4.7
- python-dateutil 2.8.1
- pytz 2021.1

- regex 2020.11.13
- requests 2.25.1
- s3transfer 0.3.4
- sacremoses 0.0.43
- sentencepiece 0.1.95
- setuptools 52.0.0.post20210125
- six 1.15.0
- tensorboardX 2.1
- tokenizers 0.7.0
- torch 1.4.0
- tornado 6.1
- tqdm 4.56.0
- transformers 2.9.0
- typing-extensions 3.7.4.3
- urllib3 1.26.3
- virtualenv-clone 0.5.4
- wheel 0.36.2
- xlrd 1.2.0
- XlsxWriter 1.3.7
- zipp 3.4.1

A.2. Description of computing infrastructure used:

GEFORCE RTX 2080 CUDA Version: 11.0

A.3. Runtime

- Training 2 to 2 and 1/2 hours.
- Inference: 3 minutes or less

A.4. Parameters

- Number of parameters in each model: BERT: 110 million parameters
- In the "Learning the Function for Combining the Embeddings and Pertubations" experiment, the use of the θ parameter adds on 2,048 parameters to the BERT model when there is a θ parameter for every token and every hidden dimension.

A.5. Hyperparameter Search

- TAVAT hyperparameters: adv init mag=0.2, adv lr=0.05, adv max norm=0.5, adv steps=2, adv train=1
- BERT hyperparameters: max sequence length 256, batch size 8, number of training epochs 10, adam epsilon=1e-08, learning rate=5e-05, weight decay=0.0
- The exact number of training and evaluation runs: 1 Training Run for each shown model performance. Evaluation is performed throughout training and at the end of each training.
- Bounds for each hyperparameter: adv steps and adv train can be all positive integer numbers adv init mag, adv lr, adv max norm can be all positive real numbers
- We restrict the bounds of our tuning of these hyperpameters to the values shown in Section 3.1
- The method of choosing hyperparameter values (e.g., uniform sampling, manual tuning, etc.) and the criterion used to select among them (e.g., accuracy) (*): Manual Tuning, F1 score

A.6. Dataset

- CoNLL 2003 Language: English
- Training set for CoNLL 2003: Number of examples: 14041
- Dev set for CoNLL 2003: Number of examples: 3250
- Test set for CoNLL 2003: Number of examples: 3453